Rapport de tp02

Organisation :

La première réunion était pendant la séance de tp 02, suite à nous recherche du tp 01on a opté pour le convertisseur pdf2txt qui était le plus adapter et pour la suite de notre travail on a commencé par choisir les langages maîtrisés par les membres de l'équipe et c 'était Python , C, Ruby .

Ensuit, on a comparé le temps d'exécution de chacun de ces trois langage.

Pour la suite de notre travail la plupart de nous échanges était via un groupe discord et des petites réunions au Ciri dans notre temps libre afin de régler les problèmes qu'on a rencontré.

Comparaison de temps d'exécution :

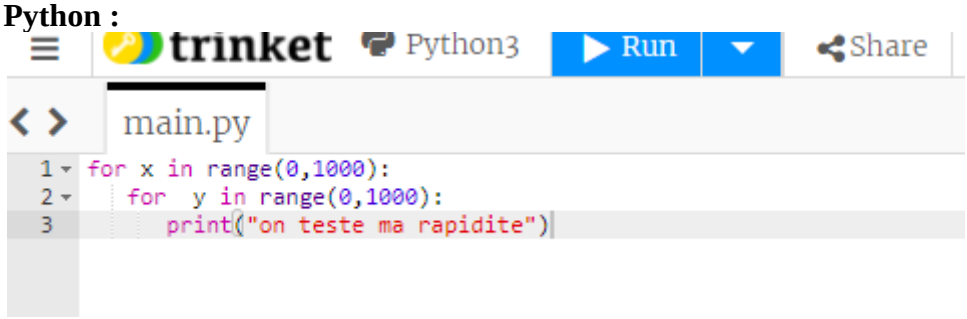
Pour comparer le temps d'exécution de chaque langage on a exécuter le même code et on a calculer le temps d'exécution .

EN langage C :

```c
#include<stdio.h>

int main (){

for (int i=0; i<1000; i++){
    for (int j=0; j<1000; j++){
    printf("on teste ma rapiditer \n");
}    public int __cdecl printf (const char * __restrict__ _Format, ...)
}

}
```

Le temps d 'exécution était de 108 secondes .

En langage Python :



Le temps d 'exécution était de 30 secondes .

En langage Ruby :

```ruby
for n in 0..1000
for m in 0..1000
  print "on teste ma rapidite\n "
end
end
```

Le temps d 'exécution était de 45 secondes .

Choix du langage :

On a remarqué qu'entre Python et Ruby il n'y avait pas beaucoup de différence en temps d'exécution ; après le vote de l'équipe on a choisi Ruby car c'était un nouveau langage pour nous qu'on voulait découvrir.

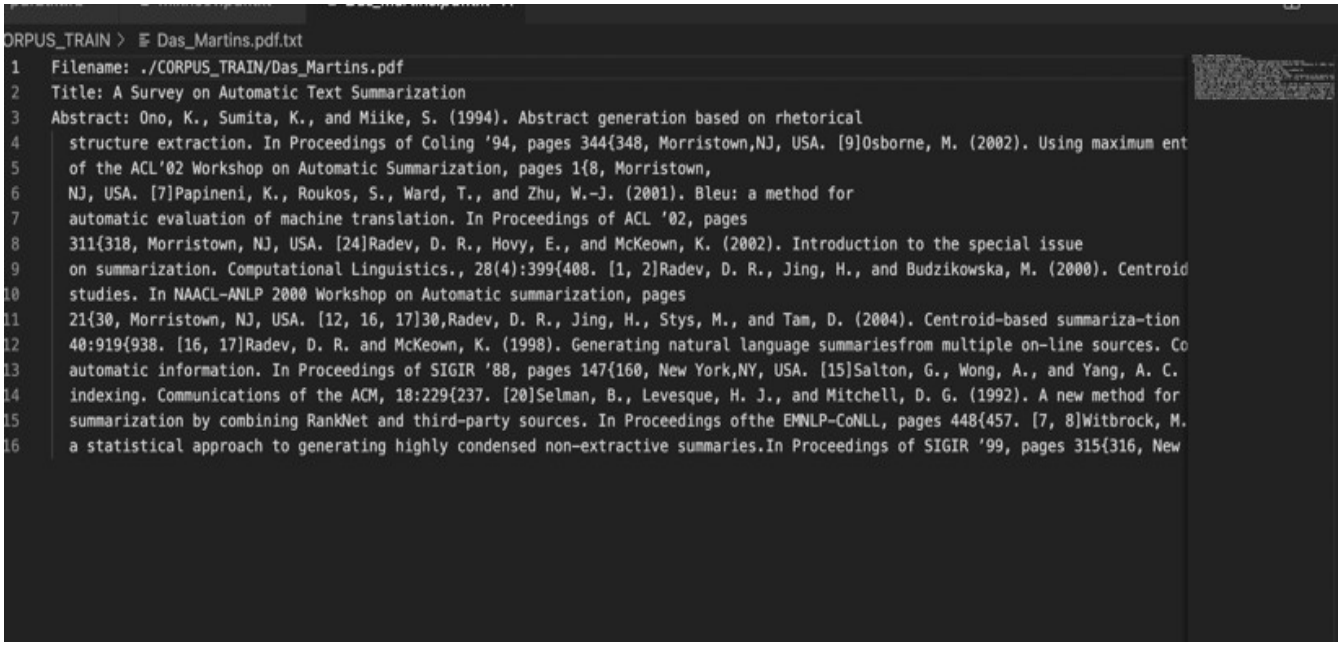
Le code en Ruby :

```ruby
#!/usr/bin/env ruby
require 'pdf/reader'

# Starting time execution
start = Process.clock_gettime(Process::CLOCK_MONOTONIC)

ARGV.each do |filename|
    # Start converting using PDF-Reader
    PDF::Reader.open(filename) do |reader|

        puts "Converting : #{filename}"
        pageno = 0
        txt = reader.pages.map do |page|

        pageno += 1
        begin
            print "Converting Page #{pageno}/#{reader.page_count}\r"
            page.text
        rescue
            puts "Page #{pageno}/#{reader.page_count} Failed to convert"
            ''
        end

    end

    puts "\nWriting text to disk"
```

```ruby
    # Spliting the text to find the Abstract
    m = txt.join(",").split("\n\n")
    m.each do |i|
        i.strip!
    end

    # Finding the Abstract if it exists its title
    i = 0
    abstractIndex = 0
    introductionIndex = 0
    while i < m.length()
        if m[i].include?("Abstract") || m[i].include?("ABSTRACT")
            abstractIndex = i
        end
        i = i + 1
    end

    # Finding the next portion of the text that might be Introduction!
    i = 0
    while i < m.length()
        if m[i].include?("Introduction") || m[i].include?("introduction")  || m[i].include?("INTRODUCTION")
        || m[i].to_s.include?("1.") || m[i].to_s.include?("I")
            introductionIndex = i
        end
        i = i + 1
    end
```

```ruby
    # Merging the distance between the Abstract title and the Introduction title to have the portion of Abstract
    abstract = ""
    i = abstractIndex
    if abstractIndex == 0
        abstract = "There is no Abstract to select!"
    elsif abstractIndex && introductionIndex
        while i <= introductionIndex
            abstract += m[i].to_s
            i += 1
        end
    end

    # Inserting to the Text file
    File.write filename+'.txt', "Filename: " + filename + "\nTitle: " + m.first.to_s + "\nAbstract: " + abstract
    end # reader

end # each


# code to time
finish = Process.clock_gettime(Process::CLOCK_MONOTONIC)

diff = finish - start # Gets time is seconds as a float

puts "Execution Time is: #{diff} seconds!"
```

Exemple de résultat :
voici un exemple du résultat obtenu à l'exécution de ce code



Explication du code :

Premièrement le fichier PDF a été convertis à une data sous forme texte.

Ensuite pour récupérer le titre, le nom et le résumé (abstract) du fichier on a divisé cette data à une table pour chercher le mots clé abstract. Et on a fait des boucles pour les mettre en ordre. à la fin, toutes les informations qui existent, ils vont être envoyer dans un fichier texte.

En plus le temps d'exécution avant et après de ce processus a été calculé.

Le temps d'exécution :

Pour chaque fichier, la moyenne du temps d'exécution était environ 2.5 seconds.

Les problèmes :

On a eu des problèmes durant la conversion des articles de deux colonnes. Par exemple on a pas pu récupérer l'abstract car il a pas bien effectué la recherche.

L'Avantage :

C'était la première fois qu'on travaille avec Ruby, la syntaxe est très simple,proche de Python et il est très rapide en exécution.